# “NAAV” Programming Language

Group 17
Team members: Atharva Date
Ansh Sharma
Vidya Rupak
Nisha Verma

## Introduction

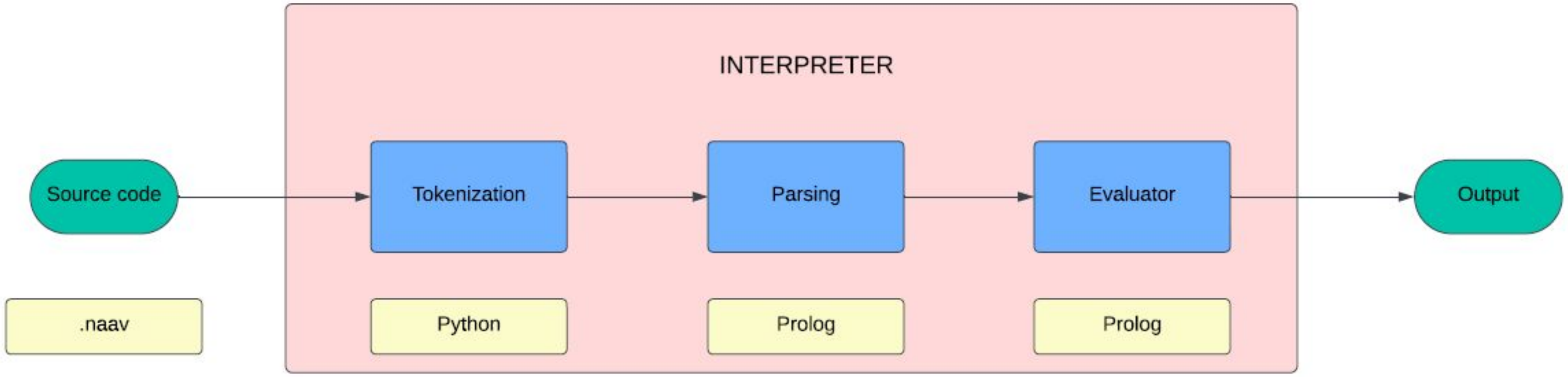
➢ Name of the language ⇒ Initials of the group members’ first name - **N**isha, **A**nsh, **A**tharva, **V**idya

➢ A versatile programming language with a user-friendly syntax ensuring code readability and ease of comprehension.

➢ It consists of familiar constructs like main methods, statement terminators, and block delimiters.

➢ It supports primitive types such as boolean, numeric, and string, along with a rich set of operators and conditional and looping constructs to deal with complex logic efficiently.

## Interpreter



## Execution

- ➢ A python file called ‘executor’ which contains the three modules in the interpreter is executed to run the NAAV code.

- ➢ The executor file is invoked via the command line, with the source code filename provided as an argument, initiating the execution process.

- ➢ The results of the code execution are presented on the command line interface, providing users with immediate feedback and visibility into the program's behavior.

## Tools and Technologies

- ➢ System: Windows OS
- ➢ Languages: Python and Prolog
- ➢ Tools:  PyCharm, SWI-Prolog, Google Drive, VSCode
- ➢ Libraries: Ply, PySwip

## Grammar

```
% Program rule
program --> block.
% Block rules
block --> [main], ["{"], declarations, [";"], commands,
printer, ["}"].
block --> commands.
% Declarations rules
declarations --> declaration, [";"], declarations.
declarations --> declaration.
% Declaration rules
declaration --> var_declaration | instant_declaration.
var_declaration --> type, id, assign_op, value.
instant_declaration --> type, id.
```

```
% Printer rule
printer --> [print], ["("], value, [")"], [";"].
printer --> [].
% Commands rules
commands --> command, [";"], commands.
commands --> command.
commands --> [].
```

## Grammar

```
% Command rules
command --> assign_value.
command --> assign_expression_value.
command --> if_else_condition.
command --> ternary_condition.
command --> increment_command.
command --> decrement_command.
command --> for_loop.
command --> range_loop.
command --> while_loop.
command --> assign_boolean_value.
command --> assign_boolean_expression_value.
assign_value --> id, assign_op, value.
assign_expression_value --> id, assign_op, expression.
if_else_condition --> [if], ["("], condition, [")"], ["{"],
block, ["}"], [else], ["{"], block, ["}"].
ternary_condition --> id, assign_op, ["("], condition,
[")"], ["?"], value, [":"], value.
increment_command --> id, [++].
decrement_command --> id, [--].
for_loop --> [for], ["("], declaration, [;], condition, [;],
increment_expression, [")"], ["{"], block, ["}"].
for_loop --> [for], ["("], declaration, [;], condition, [;],
decrement_expression, [")"], ["{"], block, ["}"].
range_loop --> [for], type, id, [in], [range], ["("], value,
[","], value, [")"], ["{"], block, ["}"].
while_loop --> [while], ["("], condition, [")"], ["{"], block,
["}"].
assign_boolean_value --> id, assign_op, value.
assign_boolean_expression_value --> id, assign_op,
boolean_expression.
```

## Grammar

```
% Expression rules
expression --> arithmetic_expression.

arithmetic_expression --> term.
arithmetic_expression --> term, arithmetic_operator,
arithmetic_expression_higher_precedence.

arithmetic_expression_higher_precedence --> term, {
precedence(Op, Precedence), Precedence > 0 },
arithmetic_operator,
arithmetic_expression_higher_precedence.
arithmetic_expression_higher_precedence --> term.

% Term rules
term --> factor | factor, arithmetic_operator, term.

% Factor rules
factor --> value.
factor --> ["("], arithmetic_expression, [")"].

% Arithmetic operator rules
arithmetic_operator --> ['/'] | ['*'] | ['+'] | ['-'].

% Type rules
type --> [num] | [bool] | [string].

% ID rules
id --> identifier.
```

## Grammar

```
% Value rules
value --> number | string | boolean | identifier.

number --> [X], { number(X) }.
string --> [X], { string(X) }.
boolean --> [T], { member(T, ['T']) } | [F], { member(F,
['F']) }.
identifier --> [X], { member(X,
[a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z]) } |
              [X], { atom(X), \+ member(X,
[a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z]) }.

% Assign operator rules
assign_op --> [=].

% Comparison operator rules
comparison_operator --> [>] | [<] | [>=] | [<=] | [==] |
[<>].
```

```
% Boolean operator rules
boolean_operator --> [and] | [or] | [not].

% Condition rules
condition --> value, comparison_operator, value |
boolean_expression.

% Boolean expression rules
boolean_expression --> value.
boolean_expression --> value, boolean_operator, value.
boolean_expression --> ["("], condition, [")"],
boolean_operator, ["("], condition, [")"].
boolean_expression --> ["("], condition, [")"],
boolean_operator, value.
boolean_expression --> value, boolean_operator, ["("],
condition, [")"].
```

THANK YOU